Freeform Search

Database: US Pre-Grant Publication Full-Text Database
US Patents Full-Text Database
US OCR Full-Text Database
EPO Abstracts Database
JPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Term:

Display: 10 **Documents in Display Format:** - **Starting with Number** 1

Generate: ○ **Hit List** ◉ **Hit Count** ○ **Side by Side** ○ **Image**

Search | Clear | Interrupt

Search History

DATE: Tuesday, June 06, 2006 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR			
L24	L23 and (rank$ or weight$)	38	L24
L23	L21 and page	97	L23
L22	L21 and page same (rank$ or weight$)	0	L22
L21	L20 and (petition or devider or section)	173	L21
L20	L19 and (garbage near3 manag$ or trash near manag$)	286	L20
L19	(modif$ or configur$ or customiz$) same (collect$ or format$)	425155	L19
L18	709/227	6363	L18
L17	709/224	9187	L17
L16	709.clas.	45070	L16
L15	711/116	41	L15
L14	711/130	796	L14
L13	711/129	957	L13
L12	711/160	368	L12
L11	711/117	953	L11
L10	711/113	2057	L10
L9	711.clas.	29867	L9

L8	707.clas.	35058	L8
L7	707/206	1298	L7
L6	707/205	2212	L6
L5	707/204	2827	L5
L4	707/103r	1771	L4
L3	707/200	4771	L3
L2	707/8	2583	L2
L1	707/3	8682	L1

END OF SEARCH HISTORY

L21: Entry 166 of 173 File: USPT Dec 1, 1998

US-PAT-NO: 5845298
DOCUMENT-IDENTIFIER: US 5845298 A

TITLE: Write barrier system and method for trapping garbage collection page boundary crossing pointer stores

DATE-ISSUED: December 1, 1998

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
O'Connor; James Michael	Mountain View	CA		
Tremblay; Marc	Palo Alto	CA		
Vishin; Sanjay	Sunnyvale	CA		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Sun Microsystems, Inc.	Mountain View	CA			02

APPL-NO: 08/841544 [PALM]
DATE FILED: April 23, 1997

INT-CL-ISSUED: [06] G06 F 17/30

US-CL-ISSUED: 707/206; 707/103, 707/104, 711/100, 711/103, 395/676
US-CL-CURRENT: 707/206; 707/104.1, 711/100, 711/103, 718/106

FIELD-OF-CLASSIFICATION-SEARCH: 707/206, 707/103, 707/104, 395/676, 711/100, 711/103, 364/140, 347/37, 347/41, 340/825.21, 434/343
See application file for complete search history.

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	4775932	October 1988	Oxley et al.	364/200
☐	4907151	March 1990	Bartlett	364/200
☐	4922414	May 1990	Holloway et al.	364/200
☐	4989134	January 1991	Shaw	364/200
☐	5088036	February 1992	Ellis et al.	395/425

☐	5136706	August 1992	Courts	395/600
☐	5218698	June 1993	Mandl	345/650
☐	5321834	June 1994	Weiser et al.	395/600
☐	5367685	November 1994	Gosling	395/700
☐	5560003	September 1996	Nilsen et al.	395/600

OTHER PUBLICATIONS

Richard Jones and Rafael Lins, Garbage Collection: Algorithms for Automatic Dynamic Memory Management, Feb. 1997, John Wiley & Sons, entire work (previously supplied copy on reserve in the EIC library) and more particulary pp. 1-41 and 116-226 (copies enclosed).
David A. Barrett, Thesis entitled: Improving the Performance of Conservative Generational Garbage Collection, Technical Report CU-CS-784-95, Sep. 1995, pp. 1-64.
Robert Courts, Improving Locality of Reference in a Garbage-Collecting Memory Management System, Communications of the ACM, Sep. 1988, vol. 31, No. 9, pp. 1128-1138.
David A. Moon, Garbage Collection in a Large Lisp System, 1984, pp. 235-246.
Urs Holzle, A Fast Write Barrier for Generational Garbage Collectors, OOPSLA '93 Garbage Collection Workshop, Oct. 1993, pp. 1-6.
David A. Moon, Architecture of the Symbolics 3600, IEEE, 1985, pp. 76-83.
Henry G. Baker, Jr., List Processing in Real Time on a Serial Computer, Comm. ACM, Apr. 1978, vol. 21, No. 4, pp. 280-294.
David Ungar, Generation Scavenging: A Non-disruptive High Performance Storage Reclamation Algorithm, ACM SIGPLAN Notices, May 1984, vol. 19, No. 5, pp. 157-167.
Guy L. Steele, Jr., Multiprocessing Compactifying Garbage Collection, Comm. ACM, Sep. 1975, vol. 18, No. 9, pp. 495-508.
Paul R. Wilson and Thomas G. Moher, Design of the Opportunistic Garbage Collector, OOPSLA '89 Proceedings, Oct. 1989, pp. 23-25.
Mario Wolczko and Ifor Williams, Multi-level Garbage Collection in a High-Performance Persistent Object System, Proceedings of the Fifth International Workshop on Persistent Object Systems, Sep. 1992, pp. 396-418.
Richard L. Hudson and J. Eliot B. Moss, Incremental Collection of Mature Objects, International Workshop IWMM 92, Sep. 1992, pp. 388-403.

ART-UNIT: 271

PRIMARY-EXAMINER: Black; Thomas Ge.

ASSISTANT-EXAMINER: Mizrahi; Diane D.

ATTY-AGENT-FIRM: Skjerven, Morrill, MacPherson, Franklin & Friel, LLP O'Brien; David W.

ABSTRACT:

Architectural support is provided for trapping of garbage collection page boundary crossing pointer stores. Identification of pointer stores as boundary crossing is performed by a store barrier responsive to a garbage collection page mask that is programmably encoded to define a garbage collection page size. The write barrier and garbage collection page mask provide a programmably-flexible definition of garbage collection page size and therefore of boundary crossing pointer stores to be trapped, affording a garbage collector implementer with support for a wide variety of generational garbage collection methods, including train algorithm type

methods to managing mature portions of a generationally collected memory space. Pointer specific store instruction replacement allows implementations that provide an exact barrier not only to pointer stores, but more particularly to pointer stores crossing programmably defined garbage collection page boundaries.

26 Claims, 10 Drawing figures

Previous Doc Next Doc Go to Doc#

L21: Entry 166 of 173 File: USPT Dec 1, 1998

DOCUMENT-IDENTIFIER: US 5845298 A
TITLE: Write barrier system and method for trapping garbage collection page boundary crossing pointer stores

Brief Summary Text (14):
An alternative to such a software barrier is to use an operating system's virtual memory page protection mechanisms to trap accesses to protected pages or to use page modification dirty bits as a map of pages potentially containing an object with an updated intergenerational pointer field. Such techniqes typically defer identifications of pointer stores, and more particularly intergenerational pointer stores, from amongst all stores until collection time. However, virtual memory page sizes are not generally well suited to garbage collection service. For example, pages tend to be large as compared with objects and virtual memory dirty bits record any modification to the associated page, not simply pointer stores. As a result the costs of scanning a page for intergenerational pointers can be high.

Detailed Description Text (13):
JAVA virtual machine implementation 250 includes hardware processor 100 and trap code executable thereon to evaluate JAVA virtual machine instructions. In addition, JAVA virtual machine implementation 250 includes hardware support for extended bytecodes (including e.g., pointer store bytecodes and memory access barriers described below in the context of garbage collection); class loader 252, byte code verifier 253, thread manager 254, and garbage collector 251 software, and microkernel 255. JAVA virtual machine implementation 250 includes a JAVA virtual machine specification compliant portion 250a as well as implementation dependent portions. Although the JAVA virtual machine specification specifies that garbage collection be provided, the particular garbage collection method employed is implementation-dependent.

Detailed Description Text (20):
FIG. 4 depicts one embodiment of a supervisor-writable register GC.sub.-- CONFIG that supports programmable filtering of stores to the heap. In the context of FIG. 1, register GC.sub.-- CONFIG, is included in registers 144 and is accessible to execution unit 140. In one embodiment, 12 bits of register GC.sub.-- CONFIG define a field GC.sub.- PAGE.sub.-- MASK for use in selecting a page size for inter-page pointer store checks. The 12 bits of field GC.sub.-- PAGE.sub.-- MASK are used as bits 23:12 of a 32-bit garbage collection page mask, with an additional 8 more-significant bits defined as 0x3F and 12 less-significant bits defined as 0x000. The resulting 32-bit garbage collection page mask is used to create a store barrier to pointer stores that cross a programmable garbage collection page boundary. Both the store data value and the objectref target of a pointer store (e.g., an aputfield.sub.-- quick instruction operating on value and objectref residing at the top of an operand stack represented at stack cache 155) are effectively masked by the 32-bit garbage collection page mask and compared to determine if value (itself an objectref) points to a different garbage collection page than that in which the target object resides. In this way, the garbage collection page size is independent of virtual memory page size. Furthermore, garbage collection pages can be provided in computer system and operating system environments, such as in low-cost, low power portable device applications or internet appliance applications, without virtual memory support. In the embodiment of FIG. 4, register GC.sub.-- CONFIG

allows programmable definition of a garbage collection page size ranging from 4 KBytes to 8 Mbytes, although, based on this description, suitable modifications for other garbage collection page sizes and size ranges will be apparent to those of skill in the art.

Detailed Description Text (46):
One embodiment of dynamic bytecode replacement is now described with reference to FIG. 7. FIG. 7 is a block diagram of a portion of a hardware processor 100 which includes an operand stack 723 which in one embodiment is represented in stack cache 155 (see FIG. 1), instruction decoder 135, non-quick to quick translator cache 131, trap logic circuit 170, software search code 31, 32 and 33 and execution unit 140. Non-quick to quick translator cache 131 includes instruction and data processor 12 and associative memory 14. Associative memory 14, in turn, includes instruction identifier memory section 18, data set memory section 20, input circuit 22 and output circuit 24.

Detailed Description Text (49):
Within associative memory 14, instruction identifier memory section 18 includes multiple (N) entries. Each of these N entries is capable of storing a corresponding bytecode identifier value, such as bytecode identifier values PC.sub.-- 0, PC.sub.-- 1, PC.sub.-- 2, PC.sub.-- 3, . . . PC.sub.-- N. Each of the bytecode identifier values stored in instruction identifier memory section 18 corresponds to a different PC value. The width of instruction identifier memory section 18 is selected to correspond with the width of the program counter.

Detailed Description Text (50):
Data set memory section 20 also includes N entries, such that each entry in instruction identifier section 18 has an associated entry in data set section 20. Each of the N entries of data set memory section 20 is capable of storing a data set, such as data sets DATA.sub.-- 0, DATA.sub.-- 1, DATA.sub.-- 2, DATA.sub.-- 3, . . . DATA.sub.-- N. As described in more detail below, each of the data sets stored in data set memory section 20 includes data for execution of the quick variant of the corresponding program occurrence of a bytecode. In one embodiment, data set memory section 20 has a width of four 32-bit words. However, data set memory section 20 can have other widths in other embodiments.

Detailed Description Text (54):
However, when the current bytecode is a non-quick bytecode having a quick variant, instruction and data processor 12 is activated in response to the current instruction. In one embodiment, bytecodes putfield and putstatic activate data processor 12. Upon activation, instruction and data processor 12 determines the status of a signal NO.sub.-- MATCH present on line 21. Initially, the instruction identifier values PC.sub.-- 0, PC.sub.-- 1, PC.sub.-- 2, PC.sub.-- 3, . . . PC.sub.-- N stored in instruction identifier memory section 18 are set to invalid values. Alternatively, `valid` bits associated with the instruction identifier values can be cleared. In either case, the current PC value provided to input circuit 22 does not initially match any of the instruction identifier values stored in instruction identifier memory section 18. Consequently, signal NO.sub.-- MATCH is asserted. The absence of a match between the current PC value and the instruction identifier values PC.sub.-- 0, PC.sub.-- 1, PC.sub.-- 2, PC.sub.-- 3, . . . and PC.sub.-- N indicates that the data set required to execute the current bytecode is not currently stored in associative memory 14. As a result, instruction and data processor 12 must initially locate and retrieve this data set to allow replacement of the non-quick bytecode with a suitable quick variant.

Detailed Description Text (60):
Instruction and data processor 12 then loads the current PC value and the retrieved data set into associative memory 14. In one example, the current PC value is written to the first entry of instruction identifier memory section 18 as instruction identifier value PC.sub.-- 0, and the corresponding retrieved data set

is written to the first entry of data set section 20 as data set DATA.sub.-- 0. The current PC value is routed from instruction and data processor 12 to memory section 18 on bus 15. The data set is routed from instruction and data processor 12 to data set memory section 20 on bus 17. The method used to select the particular entry within memory 14 can be, for example, random, a least recently used (LRU) algorithm or a first in, first out (FIFO) algorithm.

Detailed Description Text (61):
After the current PC value and the retrieved data set have been written to memory 14, instruction and data processor 12 causes the software code to retry the non-quick instruction which caused control signal TRAP to be asserted. At this time, the current PC value, which is again provided to input circuit 22, matches an instruction identifier value (e.g., instruction identifier value PC.sub.-- 0) stored within the instruction identifier memory section 18. As a result, signal NO.sub.-- MATCH is not asserted. Consequently, instruction and data processor 12 does not attempt to locate and retrieve a corresponding data set via trap logic 170 and a corresponding one of software code portions 31, 32 . . . 33.

Detailed Description Text (62):
Because the current PC value matches instruction identifier value PC.sub.-- 0, output section 24 passes corresponding data set DATA.sub.-- 0 to execution unit 140. Consequently, execution unit 140 receives the current PC value and the associated data set DATA.sub.-- 0 (including the quick variant bytecode) from non-quick to quick translator cache 131. In response, execution unit 140 executes the quick variant bytecode.

Detailed Description Text (64):
The following example will further clarify the operation of hardware processor 100, and in particular non-quick to quick translator cache 131 in facilitating a pointer-store-specific embodiment of write barrier 430 for selectively trapping pointer stores by mutator process 410 (FIG. 4). Instruction decoder 135 initially receives non-quick a bytecode (e.g., putstatic) having a quick variant, wherein the particular program occurrence of the non-quick bytecode has a corresponding PC value of 0x000100. Assuming that the particular program occurrence of bytecode putstatic is not represented in instruction identifier memory section 18, the current PC value of 0x000100 causes input circuit 22 to assert signal NO.sub.-- MATCH. In response to signal NO.sub.-- MATCH and the determination that bytecode putstatic is a non-quick bytecode having a quick variant, instruction and data processor 12 asserts control signal TRAP. Trap logic 170 uses the PC value to identify the current bytecode as bytecode INST.sub.-- 1 (i.e., putstatic). In response to the current bytecode being identified as bytecode INST.sub.-- 1, a software switch statement directs execution to corresponding software code portion 32.

Detailed Description Text (65):
Software code portion 32 then resolves constant pool entries associated with the store target object field, retrieves the data set required to execute bytecode INST.sub.-- 1, and loads this data set onto operand stack 723. Software code portion 32 provides a quick variant load bytecode to instruction decoder 135. In response, instruction decoder 135 provides a decoded quick variant load bytecode to instruction and data processor 12. Instruction and data processor 12 retrieves the data set from operand stack 723 and loads this data set into the first entry of data set memory section 20 as data set DATA.sub.-- 0. Software code portion 32 determines that the store target object field is of type reference (i.e., that the particular program occurrence of putstatic is a pointer store) and includes the appropriate pointer-specific quick variant bytecode aputstatic.sub.-- quick with data set DATA.sub.-- 0.

Detailed Description Text (66):
Instruction and data processor 12 further loads the current PC value of 0x000100

into the first entry of instruction identifier memory section 18 as instruction identifier value PC.sub.-- 0. Instruction and data processor 12 then causes non-quick bytecode INST.sub.-- 1 (i.e., putstatic) and the current PC value of 0x000100 and to be re-asserted on buses 11 and 13, respectively. In one embodiment, instruction and data processor 12 accomplishes this by issuing a return from trap (ret.sub.-- from.sub.-- trap) bytecode which transfers control back to the bytecode that caused the control signal TRAP to be asserted. At this time, input circuit 22 detects a match between the current PC value and instruction identifier value PC.sub.-- 0. In response, associative memory 14 provides the data set associated with instruction identifier value PC.sub.-- 0 (i.e., data set DATA.sub.-- 0 including the pointer-specific quick variant bytecode aputstatic.sub.-- quick) to output circuit 24. Output circuit 24 passes this data set DATA.sub.-- 0 to execution unit 140 which executes the pointer-specific quick variant bytecode aputstatic quick.

Detailed Description Text (67):
Other non-quick bytecodes having quick variants and other program instances of the same non-quick bytecode subsequently received by instruction decoder 135 are handled in a similar manner. For example, another program occurrence of the non-quick bytecode INST.sub.-- 1 (i.e., putstatic) having an associated PC value of 0x000200 can result in the PC value of 0x000200 being stored in instruction identifier section 18 as instruction identifier PC.sub.-- 1, and the data set associated with instruction INST.sub.-- 1 being stored in data set memory section 20 as data set DATA.sub.-- 1. If this particular program occurrence of bytecode putstatic resolves to a literal value store, the data set associated with instruction identifier value PC.sub.-- 1 (i.e., data set DATA.sub.-- 1) will include a quick variant bytecode such as putstatic2.sub.-- quick, rather than the pointer-specific quick variant. Note that the data set associated with the first program occurrence of non-quick bytecode INST.sub.-- 1 (e.g., data set DATA.sub.-- 0) may not be the same as the data set associated with the second program occurrence of non-quick bytecode INST.sub.-- 1 (e.g., data set DATA.sub.-- 1).

Detailed Description Text (95):
In addition, although certain exemplary embodiments have been described in terms of hardware, software (e.g., interpreter, just-in-time compiler, etc.) implementations of a virtual machine instruction processor employing various of a intergenerational pointer store trap matrix, object reference generation tagging, a write barrier responsive the intergenerational pointer store trap matrix and object reference generation tagging, a garbage collection trap handler, and/or facilities for selective dynamic replacement of pointer-non-specific instructions with pointer-specific instructions with write barrier support are also suitable. These and other variations, modifications, additions, and improvements may fall within the scope of the invention as defined by the claims which follow.

Other Reference Publication (3):
Robert Courts, Improving Locality of Reference in a Garbage-Collecting Memory Management System, Communications of the ACM, Sep. 1988, vol. 31, No. 9, pp. 1128-1138.

Previous Doc Next Doc Go to Doc#